REMARKS

Examiner Wood (hereinafter "Examiner") has rejected claims 1-9 under 35 USC § 103(a) based on two references: United States Patent No. 6,385,757 to Gupta (Issued: 07 May 2002; Priority: 20 August 1999; Assignee HP), and a printed publication entitled Specifying Representations of Machine Instructions by N. Ramsey and M. Fernandez, having listed copyright of 1997. *Examiner's Office Action* pp. 2-8 (05 December 2002).

New Claims

Applicant has carefully reviewed the art cited by Examiner. In response to Examiner, new claims have been drafted to avoid the art cited by Examiner. As explained following, such new claims are believed to be patentable over the art of record. Specifically, Applicant is herein adding by amendment new claims 10, 11, and 12. New claim 10 recites "reading an opcode summary table; analyzing said opcode summary table to determine the layout of said opcode summary table and constructing an opcode super group based on at least two opcode groups identified by said analyzing; and generating code for an instruction in architecture description language format based on said opcode super group." As pointed out by Examiner, neither Ramsey nor Gupta discuss sub-groups, *Examiner's Office Action* page 7 (05 December 2002); likewise, neither Ramsey nor Gupta discuss the "super group" related recitations in new claim 10. Furthermore, assuming *arguendo*[1] that Examiner is correct in his assertions that Ramsey applies to Applicant's claims, *Examiner's Office Action* page 7 (05 December 2002), Applicant respectfully points out that then Ramsey clearly teaches away from Applicant's "super group" recitations of claim 10. Accordingly, new claim 10 is not rendered unpatentable by the art of record.

New claim 11 recites " reading an opcode super group table; creating a plurality of output files; analyzing said opcode super group table to determine a layout of said opcode super group table; determining a presence of a sub-group from said opcode super group table; generating root code in architecture description language format based on the sub-group; cycling

[1] Applicant does not concede that Ramsey does teach this point, which is why the assumption is designated as made *arguendo*.

to generate detailed code for the sub-group in architecture description language format; repeating said cycling and determining until the end of the opcode super group table is reached; and closing said plurality of output files." As pointed out by Examiner, neither Ramsey nor Gupta discuss sub-groups, *Examiner's Office Action* page 7 (05 December 2002); likewise, neither Ramsey nor Gupta discuss the "super group" related recitations in new claim 11. Furthermore, assuming *arguendo* that Examiner is correct in his assertions that Ramsey implies sub-groups, *Examiner's Office Action* page 7 (05 December 2002), Applicant respectfully points out that then Ramsey clearly teaches away from Applicant's "super group" and/or "sub-group" recitations of claim 11. Accordingly, new claim 11 is not rendered unpatentable by the art of record.

New claim 12 recites "a first computer code section for reading an opcode summary table having a plurality of entries representative of a like plurality of microprocessor instructions; a second computer code section for producing a first grouping of at least two of said entries in accordance with a grouping criteria; a third computer code section for producing a second grouping of at least two of said entries in accordance with a grouping criteria; a fourth computer code section for producing a super grouping of the first and second grouping; and a fifth computer code section for generating an encoded representation of said super grouping.." As pointed out by Examiner, neither Ramsey nor Gupta discuss sub-groups, *Examiner's Office Action* page 7 (05 December 2002); likewise, neither Ramsey nor Gupta discuss the "super group" related recitations in new claim 12. Furthermore, assuming *arguendo* that Examiner is correct in his assertions that Ramsey implies sub-groups, *Examiner's Office Action* page 7 (05 December 2002), Applicant respectfully points out that then Ramsey clearly teaches away from Applicant's "super group" recitations of claim 12. Accordingly, new claim 12 is not rendered unpatentable by the art of record.

<u>Existing Claim Rejections</u>

In his Office Action Examiner rejected claims 1-9 under 35 U.S.C. 103(a) as being unpatentable over Ramsey et al., "Specifying Representations of Machine Instructions" in view of Gupta et al. (USPN 6,385,757). *Examiner's Office Action* page 2 (05 December 2002). At the heart of Examiner's rejection was Examiner's conclusion that the Specification Language For Encoding And Decoding (SLED) described in Ramsey was the same as or equated to the term "Architecture Descriptor Language" (ADL) as used in Applicant's claims 1-9. *Examiner's Office Action* page 3 (05 December 2002) ("Ramsey's resulting expression is in SLED an ADL.")

Applicant strongly contests that the SLED is an ADL as utilized in applicant'sApplicant's written description. In Applicant's written description, Applicant recites commercially viable subject matter as follows:

> Microprocessor designers balance many considerations during the development of a microprocessor. One important consideration is the actual functionality of the microprocessor. A microprocessor must have a plurality of instructions to be useful. The term "instruction set" is used for the set of all of the instructions utilized by microprocessor. The description of the instruction set in the functions at each instruction is called an "instruction set architecture" ("I SA"). An ISA is generally viewed in terms of three portions: the assembly language mnemonic for each of the instructions, the machine language encoding of the instructions, and the operation of the instruction.
>
> Designers traditionally describe the ISA of a processor/architecture in terms of an opcode summary table. An opcode summary table is a table which lists the assembly language mnemonic for each instruction, the machine language bit pattern encoding for each instruction, and the position of the operands within the instruction.
>
> Each bit in an instruction has a different function. The portion of the instruction that determines which instruction is to be performed is termed the opcode. For a simple instruction set, the opcode may take 6 bits out of the 32 bits available for each instruction. The operand is the remainder of the instruction and determines, for example, what register is being operated upon, or what memory location is to be accessed.
>
> When designing a new microprocessor, the designers must the able to test their designs to find any problems and optimize the performance. Testing the designs involves the use of assemblers, compilers, and simulators ("programming tools") to test the operation of the microprocessor....
>
> To automate the creation of those programming tools, several Architecture Description Languages ("ADL") have been developed, such as nML, ISDL, LISA, and RADL. It is possible to describe each instruction using an ADL. For many instruction sets, however, similar instructions have similar machine language formats with common shared fields. Using an ADL, one can group similar instructions together to create a more compact representation of the processor. This compact representation is also less prone to errors because similar groups of instructions need be described only once, with only the differences further delineated.
>
> The power of an ADL lies in the fact that, using an ADL description of a microprocessor, one can automatically create various programming tools for that processor. In order to create, for example, an assembler for a microprocessor, one may write a program in ADL that describes the microprocessor. Then, an

> assembler generator could be used to create an assembler from the ADL description of the microprocessor. In order to create a simulator, one would also need to input the behavior of each instruction into the ADL description into the simulator generator.
>
> In the past, the creation of an ADL description of a microprocessor was manually performed by a person, given the assembly language instructions and its machine language representation. However, when developing new microprocessors, the manual creation of these programs can delay the development of the new microprocessors by two or three weeks.
>
> After the programming tools are created, the new microprocessor is tested. Once tested, the design may be improved in various ways, including modifying the instruction set. Once the product is improved, new programming tools must be created, which results in additional delays associated with the re-coding of the ADL. Furthermore, the ADL code must be debugged before it can reliably be used to test the microprocessor.
>
> Another problem with the ADL is the learning curve. A microprocessor designer would have to learn how to code in a language that may be new to them. A microprocessor designer may be more concerned with producing the physical embodiment of the microprocessor chip, rather than the production of programming tools. Many designers are also more comfortable with the traditional methods of describing a microprocessor, such as an opcode summary table. Thus, many designers spend their time producing documentation related to the operation of the microprocessor.
>
> To facilitate the design of microprocessors, it is desirable to automate the process of producing an ADL representation of a microprocessor

Applicant's Application pages 2-4.

In contrast to the foregoing, Ramsey discloses as follows:

> We present SLED, a Specification Language for Encoding and Decoding, which describes abstract, binary, and assembly-language representations of machine instructions. Guided by a SLED specification, the New Jersey Machine-Code Toolkit generates bit-manipulating code for use in applications that process machine code. Programmers can write such applications at an assembly-language level of abstraction, and the toolkit enables the applications to recognize and emit the binary representations used by the hardware.

Ramsey page 492.

Applicant respectfully points out that the foregoing **is almost the exact inverse of the ADL as described in Applicant's written description**. Specifically, in Applicant's written description, Applicant states that in ADL, the system designers typically construct an opcode summary table of an actual microprocessor and then utilizes ADL to construct programming tools to simulate an actual microprocessor without building it. In contrast, Ramsey seems to teach going in the opposite direction (i.e., Ramsey teaches from the general to the more specific) of ADL as used in Applicant's written description (i.e., from the specific to the more general). However, the differences between Applicant's ADL and Ramsey do not stop there.

Specifically, in one of the portions of Ramsey identified by Examiner, Ramsey states as follows:

> Because machine instructions do not always fit in a machine word, the [SLED] toolkit works with streams of instructions, not individual instructions. An instruction stream is like a byte stream, except that the units may be "tokens" of any size, not just 8-bit bytes. An instruction is a sequence of one or more tokens, so "tokens stream" might be a more precise term.

Ramsey at page 495 bracket 1 as identified by Examiner.

Applicant points out that, as explained above, in Applicant's written description, the opcode summary table actually utilizes machine language bit level instructions, because the subject matter is in part directed toward automating a portion of the related art process by which actual microprocessors are simulated by use of ADL. Consequently, ADL represents and makes use of actual bit level machine language instructions of a microprocessor to be simulated. In contrast, as shown in the foregoing quotation from Ramsey, and as further shown on pages 496 through 498 of Ramsey, Ramsey does not teach using the actual specified machine language instructions of a microprocessor to be simulated, but rather instead teaches specifying the desired function at a very high-level of various abstract representations, and methodology to produce specific machine language level instructions in response to the high-level abstract representation. Hence, Ramsey's SLED teaches going from a high-level abstraction to specific machine language instructions, which is the exact inverse of ADL as utilized in Applicant's written description. Hence, Applicant respectfully submits that Ramsey does not in anyway show or pertain to any pending claim, and hence does not render unpatentable Applicant's claims at issue.

In addition to the foregoing, Applicant points out that, on page 494, Ramsey states using the toolkit reduces retargeting effort and makes code more reliable. For example, ldb's disassembler for the MIPS requires less than 100 lines of code, and mld has replaced 450 lines at hand-written MIPS code with generated encoding and relocation procedures. By hiding shift and mask operations, by replacing case statements with matching statements, and by checking specifications for consistency, the toolkit reduces the possibility of error. The toolkit can speed up applications that would otherwise have to generate assembly language instead of binary code." Thus, Ramsey began teaches generating the binary code instruction statements from the high-level representations, which is again the exact opposite of ADL as utilized in Applicant's written description.

As shown foregoing, Ramsey does not relate to ADL, nor opcode summary tables, as used in Applicant's claims 1-9. Accordingly, Ramsey, not being concerned with ADL, does not show or suggest at least the ADL and opcode summary related recitations of Applicant's claims 1-9.

Assuming *arguendo*,[2] that Ramsey did relate to ADL, as noted by Examiner, Ramsey does not teach automating their disclosed scheme. *Examiner's Office Action* page 3 (05 Dec. 2002). Specifically, Examiner has stated " Gupta demonstrated that it was known at the time of invention to "computerize" a method of reading opcode tables to produce a description language (Gupta: column 4, line 65 to column 5, line 27; column 3, lines 49-54). It would have been obvious to one of ordinary skill in the art at the time of invention to implement Ramsey's ADL with the automated/computerized reading of the opcode table to produce the description language code as found in Gupta's teaching." *Examiner's Office Action* page 3 (05 Dec. 2002). Applicant has reviewed the portion of Gupta cited by Examiner, and disagrees that it teaches as stated by Examiner. Specifically, the cited portion of Gupta recites follows:

> The modules in the VLIW system may be used individually or in a variety of combinations for unique VLIW processor design scenarios. In one such scenario, the datapath synthesizer takes an abstract Instruction Set Architecture (ISA) specification and programmatically generates a datapath in a hardware description language using hardware macrocells from the macrocell library. The instruction format designer then programmatically generates the processor's instruction format. The control path synthesizer may be used to construct a hardware description of the components in the processor's instruction unit.
>
> In another scenario, the system extracts the abstract ISA specification from a concrete ISA specification of a VLIW processor, including the instruction format and register files specification of the processor. By extracting the abstract ISA specification, the system may then proceed to build the datapath, extract an MDES, and build a control path based on the abstract ISA specification. The system may also optimize the instruction format using operation issue statistics to select custom templates for an application program or set of applications.
>
> In another scenario, the system extracts the abstract ISA specification from a VLIW datapath description. Using the abstract ISA, the system may then proceed

[2] Which Applicant continues to assert it does not.

to generate the instruction format, extract an MDES, and construct the control path. As above, the system may also use the MDES to select custom templates.

Gupta '757 patent col. 4 line 65 - col. 5 line 24.

Applicant respectfully asserts that the foregoing cited portions of Gupta relate to the creation of hardware, such as an ASIC, in response to a specified Instruction Set Architecture (e.g. such as by use of Verilog's HDL (hardware descriptor language)). Applicant asserts that the quoted portions of Gupta do not teach automating the scheme of Ramsey. Accordingly, assuming arguendo the Examiner is correct in his conclusion that Ramsey's SLED equates to Applicant's ADL, Applicant respectfully contests Examiner's assertion that there is a teaching to automate the scheme of Ramsey, in that the portions of Gupta identified by Examiner describe using known programming tools to create geometrical circuitry layouts. In addition, as pointed out in the portion of Applicant's written description quoted above, one aspect of the subject matter disclosed by Applicant is that Applicant recognized the need to automate the manual related art processes involved in creating an ADL representation of a specific microprocessor design, which is contra to Examiner's conclusion that such was known in the art .

As shown above, Ramsey does not relate to either ADL or the opcode summary related portions of Applicant's written description. Accordingly, Ramsey does not teach Applicant's claims 1-9. In addition, Applicant has demonstrated that, even if Ramsey did relate to Applicant's claims 1-9, there is no teaching in the arts to modify the scheme of Ramsey. Accordingly, the art of record does not render Applicant's claims 1-9 obvious.

Existing Claim Objections

In the drawings, Figures 1, 2, 7 and 8 were objected to because lines, numbers and letters are not uniformly thick and well defined, clean, durable, and black. Formal drawings correcting these deficiencies will be submitted after allowance.

Figures 1, 7 and 8 were objected to because of insufficient margins. Filed concurrently herewith are informal Figures 1, 7 and 8, which have been modified to provide sufficient margins. Applicant points out that informal drawings may be used during prosecution, so long as the informal drawings fit within the prescribed margins.

Conclusion

Overall, the cited references do not singly, or in any motivated combination and/or modification, teach or suggest the claimed features of the embodiments recited in claims 1- 11, and thus such claims are allowable. If the undersigned attorney has overlooked a relevant teaching in any of the references, the Examiner is requested to point out specifically where such teaching may be found.

In light of the above amendments and remarks, Applicants respectfully submit that all pending claims are allowable. Applicants, therefore, respectfully request that the Examiner reconsider this application and timely allow all pending claims. The Examiner is encouraged to contact Mr. Cook by telephone to discuss the above and any other distinctions between the claims and the applied references, if desired. If the Examiner notes any informalities in the claims, he is encouraged to contact Mr. Cook by telephone to expediently correct such informalities.

Attached hereto is a marked-up version of the changes made to the specification and claims by the current amendment. The attached page is captioned "**Version With Markings to Show Changes Made.**" If a conflict arises between the clean copy and the attached "**Version With Markings to Show Changes Made,**" this statement constitutes public notice that Applicants respectfully request that their intent is that the version with changes made be considered controlling.

Respectfully submitted,
Charles Paul Siska, Jr.
SEED Intellectual Property Law Group PLLC

[illegible]

Dale R. Cook
Registration No. 42,434

Enclosure:
Postcard

701 Fifth Avenue, Suite 6300
Seattle, Washington 98104-7092
Phone: (206) 622-4900
Fax: (206) 682-6031

VERSION WITH MARKINGS TO SHOW CHANGES MADE

In the Claims:

Please add new Claims 10-12 to read as follows:

1. A computerized method for producing code in an architecture description language, said method comprising the steps of:

a. reading an opcode summary table;

b. analyzing said opcode summary table to determine the layout of said opcode summary table ;

c. generating code for an instruction in architecture description language format; and

d. repeating said generating step for each line in said opcode summary table, resulting in an ADL representation of the opcode summary table.

2. The method of claim 1 where the opcode summary table is provided in a spreadsheet.

3. The method of claim 1 where the opcode summary table is provided in a comma separated value format.

4. A computerized method for producing code in an architecture description language format, said method comprising the steps of:

a. reading an opcode summary table;

b. creating a plurality of output files;

c. analyzing said opcode summary table to determine the layout of said opcode summary table;

d. determining the beginning of a group from said opcode summary table;

e. generating root code for the hierarchy in architecture description language format based on said grouping;

f. cycling through each group to generate detailed code in architecture language format;

g. repeating said cycling step until the end of the opcode summary table is reached; and

h. closing said plurality of output files.

5. The method of claim 4 where the opcode summary table is provided in a spreadsheet.

6. The method of claim 4 where the opcode summary table is provided in a comma separated value format.

7. The method of claim 4 where the opcode summary tablet is pre-formatted such that the opcodes are separated into groups prior to be read.

8. The method of claim 4 where said cycling step further comprises determining the presence of sub-groups within said group and generating detailed code for each sub-group within said group.

9. A computer program comprising:

a first computer code section for reading an opcode summary table having a plurality of entries representative of a like plurality of microprocessor instructions;

a second computer code section for producing a grouping of at least two of said entries in accordance with a grouping criteria; and

a third computer code section for generating an encoded representation of said grouping.

10. (New) A computerized method for producing code in an architecture description language, said method comprising the steps of:

reading an opcode summary table;

analyzing said opcode summary table to determine the layout of said opcode summary table and constructing an opcode super group based on at least two opcode groups identified by said analyzing; and

generating code for an instruction in architecture description language format based on said opcode super group.

11. (New) A computerized method for producing code in an architecture description language format, said method comprising the steps of:

reading an opcode super group table;

creating a plurality of output files;

analyzing said opcode super group table to determine a layout of said opcode super group table;

determining a presence of a sub-group from said opcode super group table;

generating root code in architecture description language format based on the sub-group;

cycling to generate detailed code for the sub-group in architecture description language format;

repeating said cycling and determining until the end of the opcode super group table is reached; and

closing said plurality of output files.

12. (New) A computer program comprising:

a first computer code section for reading an opcode summary table having a plurality of entries representative of a like plurality of microprocessor instructions;

a second computer code section for producing a first grouping of at least two of said entries in accordance with a grouping criteria;

a third computer code section for producing a second grouping of at least two of said entries in accordance with a grouping criteria;

a fourth computer code section for producing a super grouping of the first and second grouping; and

a fifth computer code section for generating an encoded representation of said super grouping.

341777